IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Duvdevani et al. Docket No: U 014858-1

Appln. No.: 10/706,440 Group Art Unit: 2623

Confirmation No.: 8689 Examiner: KIBLER, Virginia M.

Filed: 11/12/2003

For: APPARATUS AND METHOD FOR THE INSPECTION OF OBJECTS

Honorable Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

DECLARATION UNDER 37 CFR 1.131

Sir:

I, the undersigned, Shmuel Rippa, hereby declare as follows:

1) I am an Applicant in the patent application identified above, and I am one of the co-inventors of the subject matter described and claimed in claims 1 – 27 therein.

2) In 1996 I joined a the ICP group of the Assignee as a software engineer on a software development team assigned to develop and implement a software system performing the functionalities described and claimed in the subject Application in systems for inspecting lead frames, and eventually ball grid array substrates, being developed at the time by the Assignee. I later advanced and became team leader of the software development team.

3) Prior to February 5, 1998, our team reduced to practice the invention described and claimed in the subject application by developing working software code that was adapted to inspect ball grid array substrates. Our team did its work in Israel, a WTO country.

4) Appendix A contains a paper that I wrote on or before October 29, 1996 entitled, "SIP Tests in Windows", and stored as a computer file under the name "sip_tests_in_windows.fm*". "SIP" is an acronym for "Software Image Processing". Appendix A is part of system

documentation for a generic model of software defect detection packages implementing the invention described and claimed in the subject application.

I have printed Appendix "A" without making any modification whatsoever to the document as stored on the Assignee's computer system. The date appearing in the header of Appendix A is automatically inserted by a date field to display the date upon which the document was printed. A screen shot of the computer directory in which Appendix A is stored is contained in Appendix B and indicates that Appendix A was last modified on October 29, 1996.

5) Appendix C contains working software code, written in the "C" programming language, implementing the invention described and claimed in the subject patent application. The code in Appendix "C" was developed in accordance with the generic model documented in Appendix A. The code in Appendix "C" is part of, and runs with, a suite of software programs performing defect detection during the inspection of ball grid array substrates.

As evidenced by the revision log appearing on page 22 et seq. of Appendix C, Version 1.1 of the code is dated October 5, 1997. According to our practice, only working and reviewed versions of code were numbered. As further evidenced by the revision log, this code was regularly updated at least until August 29, 2001, well after the filing date of the subject application.

6) The following table shows the correspondence between the elements of the system claims in the present patent application and elements of the material in appendices. For reasons of simplicity and clarity of understanding, reference is made only to Appendix A, which documents a generic software defect detection package as described and claimed in the subject Application.

Claim 1	**Appendix A**
1. A system for inspecting electrical circuits comprising:	The entire document, evidenced with particularlity at the illustration on page 7, marked "A".
a boundary identifier operative to generate a representation of boundaries of elements in an image of an electrical circuit which is under inspection; and	The illustration on page 7 marked "A" shows boundaries (referred to as cels/contour elements) for the circular pad.
a defect identifier operative to receive said representation of boundaries of elements and to analyze at least some locations of at least some boundaries in said representation of boundaries of elements to identify defects in said electrical circuit.	Text at page 8 marked "B".
Claim 2	**Appendix A**
2. A system according to claim 1, and wherein said boundary identifier is operative in hardware.	Text at page 2 marked "C".
Claim 3	**Appendix A**

3. A system according to claim 1, and wherein said defect identifier is operative in software.	Text at pages 4 & 5 marked "D".
Claim 4	**Appendix A**
4. A system according to claim 2, and wherein said defect identifier is operative in software.	Text at pages 4 & 5 marked "D".
Claim 5	**Appendix A**
5. A system according to claim 1, and wherein said defect identifier is operative to compare an actual location of at least one boundary from among said boundaries in said image under inspection, to a location of a corresponding boundary in at least one reference image.	Text at pages 4 & 5 marked "D".
Claim 6	**Appendix A**
6. A system according to claim 1, wherein said boundaries comprise contours.	Illustration at page 7 marked "A".
Claim 7	**Appendix A**
7. A system according to claim 1, and wherein said system further includes a putative defect detector operative to identify	Text at page 2 marked "E", refers to errors detected by hardware.

at least some putative defects.	
Claim 8	**Appendix A**
8. A system according to claim 7, and wherein said defect identifier is operative to analyze, from among said at least some boundaries, those boundaries that are associated with said putative defects.	Text at page 2 marked "E", text at page 2 marked "C" and text at pages 4 & 5 marked "D". CELs associated with errors detected by hardware are further evaluated for defects by software.
Claim 9	**Appendix A**
9. A system according to claim 1, and wherein said system farther includes a region of interest identifier operative to identify a portion of said image of an electrical circuit as being a region of interest.	Illustration at page 7 marked "a" shows a region of interest around a circular pad.
Claim 10	**Appendix A**
10. A system according to claim 9, and wherein said defect identifier is operative to analyze, from among at least some boundaries, only those boundaries that are in said region of interest.	Text at page 2 marked "C". computation is accelerated by restricting computation to cels in windows.
Claim 11	**Appendix A**
11. A system according to claim 8, and	Text at page 2 marked "E", text at page 2

wherein said system further includes a region of interest identifier operative to identify a portion of said image of an electrical circuit as being a region of interest, and said defect identifier is operative to analyze at least one characteristic of said at least some boundaries that are in said region of interest.	marked "C" and text at pages 4 & 5 marked "D". CELs associated with errors detected by hardware are further evaluated for defects by software.

5) Claims 12 – 20 recite a method for inspecting electrical circuits with limitations similar to those of system claims 1-11. Based on the similarity of subject matter between the apparatus and method claims for inspecting electrical circuits, it can similarly be demonstrated that the invention recited in claims 11-20 was conceived and reduced to practice prior to February 5, 1998.

6) Claims 21 – 27 recite a method for fabricating electrical circuits with limitations similar to those of system claims 1-11. Based on the similarity of subject matter between the claims for the apparatus and for the method for manufacturing electrical circuits, it can similarly be demonstrated that the invention recited in claims 11-20 was conceived prior to February 5, 1998. Inspection equipment including the invention described and claimed in the subject application for the inspection of ball grid array substrates during their manufacture, was first shipped to a customer on or about July 29, 1999, and was unpacked at the customer's facility about one week later. This system was developed by a dedicated staff of over 40 full time mechanical engineers, software developers, physicists and support staff.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and conjecture are thought to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States

Code and that such willful false statements may jeopardize the validity of the application of any patent issued thereon.

[illegible]

Shmuel Rippa, Citizen of Israel
4 Rimalt St.
Ramat Gan
Israel

Date: December 12, 2004

Appendix "A"

20 October 2004
/home/asher-a/sip_tests_in_windows.fm

Orbotech Ltd.

SIP
Tests in Windows

Shmuel Rippa

1

1. Requirements

"Interesting" things usualy happens at particular locations which occupies only a small portion of the panel's area. Such are the areas near errors detected by hardware (for example, line width violation report) or by software (for example, excess/missing reports), areas around morphological events (centres, open-ends, etc.) and areas around predetermined locations which we call top down events. The top down events are defined by processing of the CAM of the board or by a special learn scan. "E"

In order to accelerate the computation and allow more complicated image understanding algorithms we try to restrict the computation to small rectangular regions (windows) on the panel rather than doing the computation on the entire panel. Data generated by hardware and software modules and that is inside the specified window is added to the data of the window containing the data (this procedure is called packing of the data into the window). Examples of data generated by hardware include: "C"

- Gray/Colour images of parts of the panel.
- CELs (Contour ELements) computed by special hardware cards.
- Morphology events (centers, open-ends, etc.) computed by hardware cards.
- More. . .

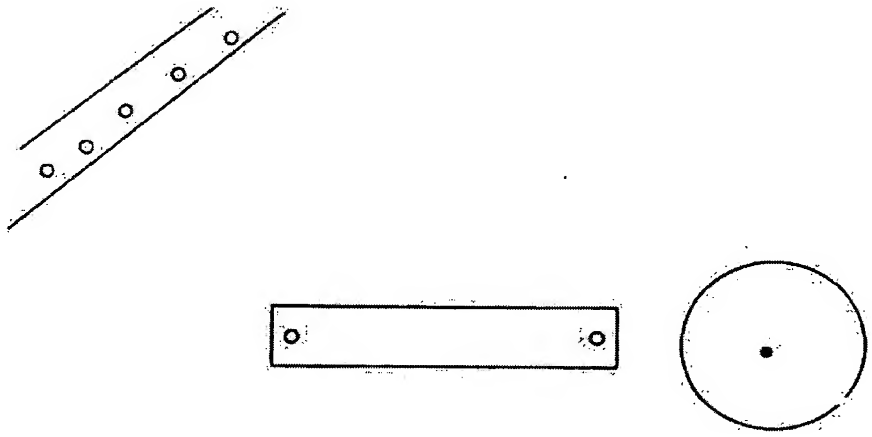
Examples of raw data genetrated by software modules include

- Excess/Missing reports
- Width violation reports.
- More . . .

Data entities are likely to be changed during the lifetime of a project as more data items are added and (perhaps) some items are removed from the above list.

In the following figure we present an example of a window containing a line segment (diagonal "fat" line), a circular shape (pad), and a rectangular shape (SMT pad) represented by CELs, two open-ends reports (displayed as dots at both ends of the SMT pad) and one center report (displayed as solid dot at the center of the circle). We also have various width violation reports (displayed as dots)on the line segment.

2



Data inside a window

Once the raw data is packed inside the window, it is ready to be processed by various algorithms. A typical algorithm operates on a window, uses the data inside the window and, perhaps, generates other data to be added to the data inside the window. In the following we give examples of some algorithms :

- Connected segment algorithm.
  The circle, rectangle and lines in the figure above are in fact represented by a collection of little line segments called CELs. These collections can be devided into four subsets. A CEL segment is in subset A iff it have a common point with another CEL from A. In the figure above we have four subsets : One for the circle, one for the rectangle and one for each line segment.
  Input : collection of CELS
  Output : subsets of conencted components ordered in a countercklockwise manner.
- Vectorizer
  Input : subset of connected components.
  Output : More compact representation of the subset (many little segments along a given line are replaced by a larger vector).
  Parameter: tolerance.
- Circle matcher. Match a circle in a least square sense to a set of points.
  Input : set of CELs
  Output: circle parameters.
- Shape finder. Searches for geometric eneties (circles, rectangles, lines, etc.). An algorithm should be written for each new geometric element that we wish to identify.
  Input : CELs or vectors from vectorizer.
  Output : Shape parameters.
  Parameters: Controling when the algorithm can safely declare on new shapoe.

The above algorithms can be used as basic blocks in higher level algorithms. For example an algorithm that searches for the presense of a circle will first devide all CELs into connected components. For each component the algorithm will decide either to vectorize the CELs or to leave them as is. In either case a call to circle shape finder will be made in order to check if the subset of conencted components contain a part of a circle. The algorithm will produce a list of circles found. The algorithm also generates the subsets of connected components and the vector representation as byproducts. The total number of data items produced by higher level algorithm can not be determined in advance since it depends in the logic flow of the algorithm. There may be several algorithms for finding circles. For example if we knew that, in a specific panel, there are isolated circles and that each circle would be identified by a center report then we could device an efficient scheme that is given the set of CELs and the centre and finds the parameters of the circle around this centre.

Requirement 1 : To be able to present, graphically, each data item and all data items produced by a higher level algorithm.

Requirement 2 : To be able to select the algorithm to use, from a library of algorithms, from a configuration file and to supply parameters for the algorithm from that file. An example of such a parameter is a parameter that specifies how much CELs have to be sufficiently close to the circle that the algorithm found in order to decide that this is realy a circle.

Requirement 3 : To be able to add data items and algorithms without changing existing code.

Requirement 4 : Reusability of algorithms.

Result : We should be able to device a TCL/TK tool that displays the raw data and then allows us to select an algorithm and to specify the parameters for that algorithm. Then we apply the algorithm and can view all the data items produced by it.

nice to have : The ability to chain several simple and higher level algoriothms to form another algorithm. For example once we wrote an algorithm for finding circles and an algorithm for finding rectangles, we could create an algorithm for finding circles and rectangles. Ideally the algorithm should be implemented without the need for recompilation or relinking by using a scripting language (TCL). The scripting language should allow us to do some computations. For example checking that the centre of the circle is at a prescribed duistance from the centre of the rectangle. We want to be able to display, graphically, all data items produced by such a new algorithm.

The learning stage:

In this stage we know very little about the scanned panel. We might have information that the panel is of a specific types (for example, it include only circles and rectangles). The data generated by hardware consists of CELs and features (centres and open-ends). We open a window around each feature. If the feature is a center, then we apply an algorithm that searches for a circle and if the feature is an open-end then we apply an algorithm that searches for a rectangle. The output of this stage is a reference file consisting of all circles and rectangles found in the panel. The reference may also include the original data CELs and features.

Requirement 5 : To be able to specify from a config file, the type of panel and thus the exact behaviour of the learning stage. We have to specify what algorithm to apply for each trigger and to specify what data items should be stored as reference.

The inspect stage:

In this stage we are given the reference data created during the learn stage. For each ref data we attach a test function which will compare the raw data in the against the reference and decide if there is an error. The test function will be given all the } "D" parameters required for making the comparison. For example a test for a ref data consisting of a circle might consist of computing the circle from the incoming data (CELs and features) and notify on error if the radius of the computed circle is different by more than a prescribed tolerance from the radius of the circle in the reference circle. Treatment of hardware/software errors is similar to the treatment in the learn stage. "D"

Requirement 6: To be able to attach, from a config file, a test function for each item in the ref file and to be able to specify the parameters of the test function from the config file.

Requirement 7: To be able to view reference data agains raw data and to apply algorithms from TCL/TK tool.

2. Sipdata (sipdata.C, sipdata.H)

This is the base class for all data items used in window related algorithms. The class uses a factory (or virtual constructor) pattern, see also documentation in file sip_factory.H, so that derived classes can be added without need to change existing code. Each object derived from Sipdata have a type which is the name of the derived class and a name which uniquely identifies the object among many objects of the same type. The function `SetName` allows to set a name of a `Sipdata` object while the function Name returns that name. The base class Sipdata do not have a default constructor. It is necessary to specify a name for the object when constructing it. The Sipdata base class provides two more public methods, `Read` and `Write` for reading and writing of `Sipdata` objects. The `Write` method use `Io_util` object to write data in ASCII or binary forms. It writes the name of the derived class and then call `DoWrite` to let derived class do the rest of the job.Reads object from file in ascii or binary mode. The `Read` method function will be called after the calling program have read the line in the file containing the name of the derived class and called the `DoCreate` function to create an instance of the derived object. That `Read` method of that object is then invoked. Calls virtual `DoRead` function to do the actual reading.

Deriving from Sipdata:

In order to derive from Sipdata it is necessary to defined the following virtual functions:

- `Index`.
  The factory pattern requires that the base class will manage a table with an entry for each derived class. Index returns the index of the derived class of a specific object in that table.
- Type.
  Returns the name of the derived class.
- `DoCreate`.
  Allocate a new object from derived class and returns a (base class_ pointer to it
- `Display`
  Use a Display_data object to display graphically the concrete data of derived class.
- `Erase`
  Clears all data and leaves an empty object.
- `DoWrite`. Used by public method `Write` of base class.
  Writes object using `Io_util` object in ascii or binary mode. W
- `DoRead`
  Reads object written by `DoWrite` from file in ascii or binary mode.

The base class was designed so that it is possible to add new derived class in a very simple way and without changing the code that is responsible for creation of derived objects stored in a file. To derive a class one have to folllow the instructions in `sipdata.H` file. Then all that is needed is to link the new code with the application.

Subclasses that have been derived so far are

Table 1: Subclassed derived from Sipdata

File names	Description
sipdata_cel_events.C sipdata_cel_events.H	collection of CELs
sipdata_feature_events.C sipdata_feature_events.H	collection of features
sipdata_connected_components.C sipdata_connected_components.H	conencted components decomposition of CELs
sipdata_isolated_circles.C sipdata_isolated_circles.H	All circles defined by a single, closed, connected component.

3. Rect_win (rect_win.H), Sipwin (sipwin.C and sipwin.H), Sipwin_file_of (sipwin_file_of.C and sipwin_file_of.H)

A `Rect_win` structure defining a rectangular window by specifying its endpoints `x0`, `y0`, `x1` and `y1` and its "hot point" `(xh, sub_pixel_x)`, `(yh, sub_pixel_y)` with is given in sdubpixel coordinates. The class `Sipwin` is derived from `Rect_win` and is used as a container for `Sipdata` objects.

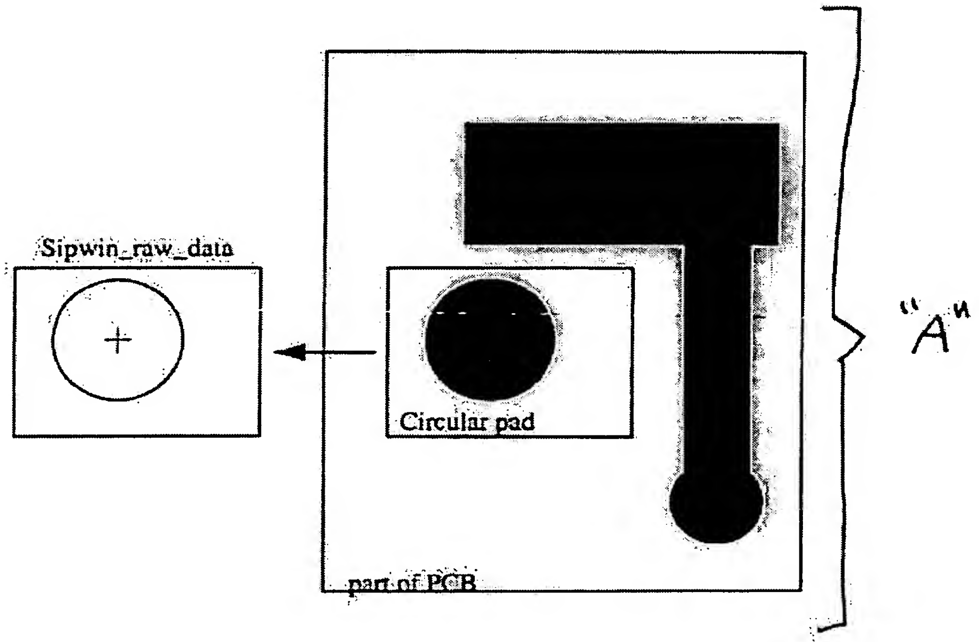
Th public methods of `Sipwin` are:

- `Put( const Sipdata * s)`.
  Put s in `Sipwin`. This represents a transfer of ownership from the caller to the window. The `Sipdata` object s is given a `const` attribute.
- `PutNonConst( const Sipdata * s )`
  The same as `Put` but s is given a non `const` attribute.
- `const Sipdata * Withdraw( int i )`
  `const Sipdata * Withdraw( const char * name )`
  Take an element out from `Sipwin`. This represents a transfer of ownership from `Sipwin` to the caller. The element can be retrieved by its index or by its name. If element not found or if it was found but its attribute is non `const`, then the method returns NULL. In addition we have two methods for taking non `const` elements out of Sipwin. These methods return NULL if element not found or if it was found but its attribute is `const`:
  `Sipdata * WithdrawNonConst( int i )`
  `Sipdata * WithdrawNonConst(const char * name )`
- `const Sipdata * Get( int i )`
  `const Sipdata * Get( const char * name )`
  Get an element from `Sipwin`. The element can be retrieved by its index or by its name. If element not found or if it was found but its attribute is non `const`, then the method returns NULL. In addition we have two methods for getting non `const` elements out of `Sipwin`. These methods return NULL if element not found or if it was found but its attribute is

```
const;
Sipdata * GetNonConst( int i ).
Sipdata * GetNonConst(const char * name )
Read.
```
Reads object from file in ASCII or binary form.
```
Write.
```
Writes object to file in ASCII or binary form.
```
Display.
```
Use Display_data object to display graphically the data inside the window.

The `Sipwin_file_of` class is responsible for reading and writing files containing `Sip-windows` from and into files. A file containing collection of `Sipwin` objects can be openned for reading or writing by using the `Open` method. The file can be closed using the `Close` method. The `NumberOfSipDataObjects` method returns the number of `Sipwin` objects in the file. The `Write` method appends a `Sipwin` object to the file and the `Read( int i )` method reads the i'th `Sipwin` object from the file.

The `Sipwin_vector_of` handles a vector containing many `Sipwin_data` elements which are stored in memory. The class provides methods for reading/writing from file (using the `Sipwin_file_of` class) and for fast random access to its entries.



3.1 `Sipwinfunc`, and `Sipwin_vector_of_functions`.

This is the base class for all functions that work on Sipwindows. The class uses a factory (or virtual constructor) pattern, see also documentation in file sip_factory.H, so that derived classes can be added without need to change existing code. Each object derived from `Sipwinfunc` have a type which is the name of the derived class and a name which uniquely identifies the object among many objects of the same type. The function `SetName` allows to set a name of a `Sipdata` object while the function `Name` returns that name. The base class `Sipwinfunc` do not have a default constructor. It is necessary to specify a name for the object when constructing it.

Each concrete function contains parameters controling its behaviour. For example, a function that checks circular pad have a parameter specifying the max allowed difference between the radius of the computed pad and the radius of the expected pad. The function will be called with a Sipwin object containing a reference circle and the CELs inside the window. The function will try to fit a circle to given CELs and reports on defect the computed radius is smaller that the expected radius minus tolerance or larger that the expected radius plus tolerance. The function will compute a description of the computed circle and add it to the data of Sipwin.p The base class will contain the following public virtual functions:

"B"

- `Type.`
  Returns the name of the derived class.
- `DoCreate.`
  Allocate a derived class object as a pointer to Sipwinfunc.
- `SetParams.`
  Sets parameters for the function from a configuration object. NOte: Each concrete function will have another, overl;oaded, version of SetParams for explicitly setting the parameters for that function.
- `Execute.`
  Perform the computation of the function given a `Sipwin` function as input. Output of computation will be added to this `Sipwin` object.

The class **`Sipwinfunc_vector_of`** is responsible to read a set of functions defined in a configuration file to provide access to the functions etc.

4. Top down methodology, Sipwin top down_reference, `TopDown` data item, top down `Sip_event`, and `TopDown` task and the class `Sipwin_top_down_tests`

The top down methodology is based on constructing a reference describing the entities to be checked. The reference contains information about the entity (for example, A circular pad of radius of 30 MIL is expected in a given location) and about the test that is to be performed (check that the radius of the pad is not smaller that 25 MIL and not larger than 37 MIL).The creation of a top down reference is very important and may be a difficult task. The reference may be created using accurate CAM data, or it may be learned from a special learn scans. Once a good top down reference is available, it can help considerably to increase the detection and to decrease the amount of false alarms.

The top down reference consists of three files :

- A file containing `Sipwin_data` objects. This file is created by a special learn scan or by processing of CAM data.
- A file containing definiitions of all functions used in the scan with their parameters. For example, if we want to fit circle to CELs with two tolerances then we will have two function definitions in the file.

A file containing a list of function names. The i'th entry in that file contains the name of the function that should be attached to the i'th window from the window file.

A TopDownReport data item consists 32 bits devided as follows:

4 bits	4 bits	24 bits
x_subp	y_subp	index in the array T

A Top_down_event class is derived from Sip_event class and contains a single Top-DownReport object.

The TopDown task loads the top down reference and is responsible for producing the top down data source. If the task works without registration than the TopDown data source consists of top down events which will allow access to the top down table. If there is a registration then the task transform the top down reference from reference coordinates to on-line coordinates and produce the top down data source.

The class Sipwin_top_down_reference contains the complete top down reference which consists of a vector of Sipwin_test_raw objects, a vector of Sipwin_test_compound objects, a referenbce table and a top down data source.

The class provides the following public methods
- LoadConfiguration.
  -- Use Sipwin_vector_of method to load all windows in the file of Sipwindows.
  -- Use Sipwinfunc_vector_of to load all functions, with their parameters, from file of functions.
  -- Run over file of function names. Buld a vector of pointers to Sipwinfunc objects from Sipwinfunc_vector_of such that the i'th entry corersponds to the function assotiated with the i'th Sipwindow.
  -- Construct a full Ds_array<Top_down_events>.
  -- Consider Meta windows. Meta window is a windor that contains data produced by functions operating on other windows. The meta window hirarchy is determined when learning the board. We define to each window, its consumers (if any) and build a data srtucture for defining the consumers of the data produced by the top down function operating on each window. In the meta window itself we shall store information on how many data items it uses.
- Clear. Undo LoadConfiguration.
- Yiterator.
  Returns iterator to the y'th line of Ds_array<Top_down_events>.
- Sipwinpack * HandleTrigger(const Top_down_event & trigger).
  The trigger is a top down event , we get the index of the corresponding (the index element of the top down item) Sipwin element in Sipwin_vector_of. The coordinates of ????? its window are transformed from the reference coordinate system to the on-line system. Returns NULL on error.

5. Sipwin_trigger_handler

This class is for handling triggers. It consist of the following public methods:

- `LoadConfiguration.`
  Load all information needed for computation.
- `Clear`. Undo LoadConfiguration.

-
  ```
  Sipwinpack * HandleTrigger(const Sipwin_top_down_reference & td_ref,
                             const Sip_event * trigger)
  ```
  If the trigger is a top down event ,call `td_ref.HandleTrigger( Top_down_event *)trigger )`. In any other case, use internal logic to create and return required data.

6. Sipwinpack

The class `Sipwinpack` helps that packer program to get information on window in wich data is packed and provide push_back functions to add packed items into Sipdata containers.

The class provides the folowing public methods:

- `Sipwinpack( int x0 int y0, int x1 int y1 )`
- IsOk()
- Win()
- push_back(...)
- NotifyAllFeaturesPacked
- NotifyAllCelsPacked

7. Top_down_ task

Is attached to transformation data source and to static top down reference data source. Produces data source of top down events after transformation and ordering of top down events in online liens. Produces alligned top down events.

8. Packer task

Attached to data sources that produce triggers, to top down data source, to cel data source to static top down reference data source and to stating trigger handler data source. Produces queue of windows data source.

The task puts all trigger data into Sip_event lists ordered by priority. Then call trigger handler for each trigger that is not inside existing window. Gets pointers to Sipwinpack objects that are inserted into list of overlapping windows. Packe triggers, data sources and CELs inside Sipwinpack objects containing them. After pack is completed, notify the Sipwinpack object on end of packing and push packed window into queue.

9. test manager task

Uses win_queue data source and static top down reference data source. It pops window from queue, execute the corresponding function and distribute results to costumers. Process defects if found and push new test into queue if neccessary.

10. `Sip_defect` and `Sip_vector_of_defects`

This class will contain description of defects detected during scan. The class is TBD. Probably it will consist of the following
- version number (static attribute of the class)
- on-line --> reference registration (Identity for learn scans)
- Location (x,y) in on-line coordinate system.
- TCL string describing the defect.

The class will provide the following public methods:
- Read/Write to and from file in TCL forms.
- Set/Get for all non static elements.

The class `Sip_vector_of_defects` will allow read/write of TCL file containing defects and provides iterator to all defects.

Appendix "B"

```
                                   icpDirectory
clyde:~> ls -ltr Icp/Dvlp/alfi/sip/Doc/
total 2076
-rwxr-xr-x    1 shmulik  ws          66560 Oct 29  1996 sip_tests_in_windows.fm*   ←
-rwxr-xr-x    1 shmulik  ws          69632 Aug 21  1997 sip_defect_detection.fm*
-rwxr-xr-x    1 shmulik  ws          92160 Oct 27  1997 sip_future_work.fm*
-rwxr-xr-x    1 shmulik  ws          28672 Oct 27  1997 sip_milestone_aug97.fm*
-rwxr-xr-x    1 shmulik  ws         113664 Oct 27  1997 sip_status.fm*
-rwxr-xr-x    1 shmulik  ws         200704 Jan  5  1998 sip_uguide.fm*
-rwxr-xr-x    1 shmulik  ws         183296 Feb  3  1998 sip_overview.fm*
-rwxr-xr-x    1 shmulik  ws          68608 Feb  3  1998 sip_navigator.fm*
-rwxr-xr-x    1 shmulik  ws          51200 Feb  3  1998 sip_config.fm*
-rwxr-xr-x    1 shmulik  ws           2055 Mar 15  1998 vectorizer_perf*
-rwxr-xr-x    1 shmulik  ws         193262 Mar 15  1998 vec_defl.tif*
-rwxr-xr-x    1 shmulik  ws         203356 Mar 15  1998 vec_maxcomp.tif*
-rwxr-xr-x    1 shmulik  ws         345926 Mar 15  1998 vecperf.tif*
-rwxr-xr-x    1 shmulik  ws            782 Mar 15  1998 vecprf2*
-rwxr-xr-x    1 shmulik  ws             29 Mar 15  1998 vectorizer_rec*
-rwxr-xr-x    1 shmulik  ws             56 Mar 15  1998 vectorizer_rec1*
-rwxr-xr-x    1 shmulik  ws           5389 Mar 30  1998 connectpcb.gif*
-rwxr-xr-x    1 shmulik  ws           6605 Mar 30  1998 connectpcbd0.gif*
-rwxr-xr-x    1 shmulik  ws           6683 Mar 30  1998 connectpcbd1.gif*
-rwxr-xr-x    1 shmulik  ws          66560 Mar 30  1998 connected.fm*
-rwxr-xr-x    1 shmulik  ws          89088 Mar 30  1998 vectorizer.fm*
-rwxr-xr-x    1 shmulik  ws          31744 Nov 15  1998 error_handling_messages.fm*
-rwxr-xr-x    1 shmulik  ws          50176 Jan 19  1999 logger.fm*
-rwxr-xr-x    1 shmulik  ws           3070 Feb 17  1999 install_instructions.txt*
-rwxr-xr-x    1 shmulik  ws          39936 Jul 29  1999 td_reference.fm
-rw-r--r--    1 shmulik  ws          53248 Dec 29  1999 bath_registration.fm
-rw-r--r--    1 shmulik  ws          54272 Dec 29  1999 mechainc_test.fm
-rw-r--r--    1 shmulik  ws           4096 Jan  8  2001 html/
drwxr-xr-x    3 shmulik  ws           4096 Jan  8  2001 CVS/
drwxr-xr-x    2 shmulik  ws
clyde:~>
```

Appendix "C"

```
#include <cmath>
#include <cstdio>
using namespace std;

#include "func_polylines_comp_aaa.H"

#include "sip_logger.H"
#include "sip_config.H"
#include "single_camera.H"
#include "dpoint2.H"
#include "gcalc.H"
#include "dseg_envelope.H"
#include "poly_compare.H"
#include "cel_event.H"
#include "cel.H"
#include "sip_factory.H"
#include "sipdata_sip_defects.H"
#include "sipdata_connected_components.H"
#include "sipdata_cel_events_raster.H"
#include "connected_component.H"
#include "sipdata_polylines.H"
#include "sipdata_segments_info.H"
#include "sipdata_winref.H"
#include "sipdata_pim_envelopes.H"
#include "pim.H"

// Add a factory for Sipdata_cel_events to list of factories for Sipdata
// define index of derived class in the vector of derived classes
//----------------------------------------------------------------------------
static Register_subclass<Func_polylines_comp_aaa> r_Func_polylines_comp_aaa;

Func_polylines_comp_aaa::Func_polylines_comp_aaa()
  : Sipwinfunc("Func_polylines_comp_aaa")
{}

Func_polylines_comp_aaa::~Func_polylines_comp_aaa()
{}

const char * Func_polylines_comp_aaa::Type() const
{
  return "Func_polylines_comp_aaa";
}

Base_factory * Func_polylines_comp_aaa::DoCreate() const
{
  return new Func_polylines_comp_aaa;
}

// Set parameters for function object from config object
//----------------------------------------------------------------------------
bool Func_polylines_comp_aaa::DoSetParams( Sip_config &  config,
                                 const Sipwinfunc_vector_of & )
{
  string mr_mode;
  if ( !(config.GetFieldElement("micro_reg_mode",mr_mode ))) {
    mlog(LOG_DERIVED_ERROR,  "DoSetParams: "
        "Function %s : Failed to get perform_micro_reg from configuration.",
       Name());
  return false;
}
if ( mr_mode == "micro_reg_with_envelopes" ) {
  micro_reg_mode = MR_ENVELOPES;
}
else if ( mr_mode == "micro_reg_icp" ) {
  micro_reg_mode = MR_ICP;
}
else if ( mr_mode == "micro_reg_none" ) {
  micro_reg_mode = MR_NONE;
}
else {
  mlog(LOG_APPLIC_ERROR,
      "Function %s: micro_reg_mode (%s) is not one of the legal mr modes"
      " (micro_reg_with_envelopes, micro_reg_icp, micro_reg_none). ",
      Name(), mr_mode.c_str());
  return false;
}

vector<string> c_mode;
if ( !(config.GetFieldElement("compare_mode",c_mode )) ) {
  mlog(LOG_DERIVED_ERROR,  "DoSetParams: "
      "Function %s : Failed to get compare_mode from configuration.",
      Name());
  return false;
}

if ( c_mode.size() == 0 ) {
  mlog(LOG_APPLIC_ERROR,"DoSetParams: "
      "Function %s : No data for parmeter compare_mode:",
      Name());
  return false;
}

compare_mode.clear();
compare_mode.reserve(c_mode.size());

bool comp_to_env        = false;
bool comp_to_pim        = false;
bool comp_to_bitmap     = false;

for ( unsigned int i = 0; i < c_mode.size(); ++i ) {
  if ( c_mode[i] == "compare_to_envelopes" ) {
    if ( comp_to_env ) {
    mlog(LOG_WARN,
          "DoSetParams: String compare_to_envelopes appears more than once.",
          Name());
    }
    else {
    compare_mode.push_back(COMPARE_CELS_2_ENVELOPES);
    comp_to_env              = true;
    }
  }
  else if ( c_mode[i] == "compare_to_bitmap" ) {
    if ( comp_to_bitmap ) {
    mlog(LOG_WARN,
          "DoSetParams: String compare_to_bitmap appears more than once.",
          Name());
      }
      else {
      compare_mode.push_back(COMPARE_CELS_2_BITMAP);
      comp_to_bitmap  = true;
      }
    }
    else if ( c_mode[i] == "compare_to_pim" ) {
      if ( comp_to_pim ) {
      mlog(LOG_WARN,
           "DoSetParams: String compare_to_pim appears more than once.",
           Name());
      }
      else {
      compare_mode.push_back(COMPARE_CELS_2_PIM);
      comp_to_pim  = true;
      }
    }
    else {
      mlog(LOG_APPLIC_ERROR,
          "Function %s: compare mode (%s) is not one of legal comparison modes"
          " (compare_to_pim, compare_to_bitmap, compare_to_envelopes). ",
          Name(),c_mode[i].c_str());
      return false;
    }
  }


  // get search width (in pixels)
  if ( !(config.GetFieldElement("search_width", search_width ))) {
    mlog(LOG_DERIVED_ERROR,  "DoSetParams: "
        "Function %s: Failed to get search_width from configuration.",
        Name());
    return false;
  }
  search_width = fabs(search_width);

  // get weights for each segment id. Used within the microreg process.
  micro_reg_segment_weight.clear();
  if (
!(config.GetFieldElement("micro_reg_segment_weight",micro_reg_segment_weight )))
{
    mlog(LOG_DERIVED_ERROR,  "DoSetParams: "
        "Function %s : Failed to get micro_reg_segment_weight "
        "from configuration.", Name());
    return false;
  }
  if ( micro_reg_segment_weight.size() == 0 ) {
    mlog(LOG_APPLIC_ERROR,"DoSetParams:"
        "Function %s : No elements in vector micro_reg_segment_weight ",
        Name());
    return false;
  }

  //-----------------------------------------------------------------------
  // read  list of requested width defects (up and down directions).
  // In j'th location of this list we
  // will find the defect width tolerance corresponding to segment having id
  // that is equal to j.
```

```
//------------------------------------------------------------------------
defect_width_up.clear();
if ( !(config.GetFieldElement( "defect_width_up",defect_width_up) ) ) {
  mlog(LOG_APPLIC_ERROR, "DoSetParams: "
      "Function %s: Can not read list defect_width_up.", Name());
  return false;
}

if ( defect_width_up.size() != micro_reg_segment_weight.size() ) {
  mlog(LOG_APPLIC_ERROR, "DoSetParams: "
      "Function %s: Unequal Number of elements in lists defect_width_up "
      "and micro_reg_segment_weight.",
      Name());
  return false;
}

defect_width_down.clear();
if ( !(config.GetFieldElement( "defect_width_down",defect_width_down) ) ) {
  mlog(LOG_APPLIC_ERROR, "DoSetParams: "
      "Function %s: Can not read list defect_width_down.", Name());
  return false;
}

if ( defect_width_down.size() != defect_width_up.size() ) {
  mlog(LOG_APPLIC_ERROR, "DoSetParams: "
      "Function %s: Unequal Number of elements in lists defect_width_up "
      "and defect_width_down.",
      Name());
  return false;
}

defect_tang_width.clear();
defect_tang_width.reserve(defect_width_up.size());
for ( unsigned int i = 0; i < defect_width_up.size(); ++i ) {
  double tang_width = ( defect_width_up[i] + defect_width_down[i] ) / 2.0;
  defect_tang_width.push_back(tang_width);
  if ( i == 0 ) {
    max_tang_width = defect_tang_width[i];
  }
  else {
    if ( max_tang_width < defect_tang_width[i] ) {
    max_tang_width = defect_tang_width[i];
    }
  }
}

if ( !(config.GetFieldElement("angle_tolerance_for_micro_reg",
                        &mr_angle_tolerance ))) {
  mlog(LOG_DERIVED_ERROR, "DoSetParams: "
      "Function %s: Failed to get angle_tolerance_for_micro_reg from
configuration.",
      Name());
  return false;
}
mr_angle_tolerance = fabs(mr_angle_tolerance);

if ( !(config.GetFieldElement("angle_tolerance_for_shape_compare",
                        &sc_angle_tolerance ))) {
```

```
    mlog(LOG_DERIVED_ERROR, "DoSetParams: "
        "Function %s: Failed to get angle_tolerance_for_shape_compare "
        "from configuration.", Name());
    return false;
  }
  sc_angle_tolerance = fabs(sc_angle_tolerance);

  // Define vector of angle tolerances for each segment Id
  v_sc_angle_tolerance.clear();
  v_sc_angle_tolerance.resize(defect_width_up.size());
  for ( unsigned int i = 0; i < defect_width_up.size(); ++i ) {
    v_sc_angle_tolerance[i] = sc_angle_tolerance;
  }


  if ( !(config.GetFieldElement("small_cel", small_cel ))) {
    mlog(LOG_DERIVED_ERROR, "DoSetParams: "
        "Function %s : Failed to get small_cel from configuration.", Name());
    return false;
  }
  small_cel = fabs(small_cel);

  if ( !(config.GetFieldElement("max_number_excess_cels",max_number_excess_cels
)))
  {
    mlog(LOG_DERIVED_ERROR, "DoSetParams: "
        "Function %s: Failed to get max_number_excess_cels  "
        "from configuration.", Name());
    return false;
  }

  if (
!(config.GetFieldElement("max_number_empty_envelopes",max_number_empty_envelopes
)))
  {
    mlog(LOG_DERIVED_ERROR, "DoSetParams: "
        "Function %s: Failed to get max_number_empty_envelopes  "
        "from configuration.", Name());
    return false;
  }

  //--------------------------------------------------------
  // 0 transmissive 1 reflective
  if ( !config.GetFieldElement("polarity", polarity )) {
    mlog(LOG_DERIVED_ERROR, "DoSetParams: Function %s: "
        "Failed to get polarity field from configuration",
        Name());
    return false;
  }


  //--------------------------------------------------------
  if ( !config.GetFieldElement("treat_all_pinholes_as_surface_defects",
                               treat_all_pinholes_as_surface_defects )) {
    mlog(LOG_DERIVED_ERROR, "DoSetParams: Function %s: "
        "Failed to get treat_all_pinholes_as_surface_defects field from
configuration",
        Name());
```

```
      return false;
    }

    return true;
  }

bool Func_polylines_comp_aaa::DoClear()
{
  defect_width_up.clear();
  defect_width_down.clear();
  defect_tang_width.clear();
  micro_reg_segment_weight.clear();
  return true;
}

//-----------------------------------------------------------------------------------------
// Execute microregistration and comparison step
//
// Microreg options
// ----------------
// MR_ENVELOPES                    : Perform micro registration by searching for
online
                                     CELs within envelopes of reference polylines.
//                                   The envelopes are created after ref. polylines
//
are
                                     transformed to online coordinates.
//
// MR_ICP                          : Perform MR by matching to each segment of ref
polyline
                                     a corresponding point on online CEL.
//
// MR_NONE                         : No micro registration.
//
//
// Comparison options
// ------------------
// COMPARE_CELS_2_ENVELOPES        : Compare CELs to envelopes of polylines
//                                   polylines transformed to online coordinates.
// COMPARE_CELS_2_BITMAP           : Compare CELs to bitmap of envelopes of polylines.
//                                   Polylines are transformed to online coordi
//                                   and then envelopes are created.
// COMPARE_CELS_2_PIM              : Compare CELs to PIM representation of envelopes
//                                   CELs are transformed to reference coordinates and
//                                   checked against PIM of envelopes.
//-----------------------------------------------------------------------------------------
Sipwinfunc::Return_code
Func_polylines_comp_aaa::Execute( Sipwin & w ) const
{
  mlog(LOG_STANDARD, "Execute: "
       "Function %s: Processing window %s(%d)  [%d %d]x[%d %d]",
       Name(), w.Type(), w.Id(),w.X0(), w.X1(), w.Y0(), w.Y1() );

   //--------------------------------------------------------------------------------------
   // Extract and allocate data needeed for computation
   //--------------------------------------------------------------------------------------
   Func_polylines_comp_aaa::DataForCompare data_for_compare;
   switch ( ExtractDataForCompare( w, data_for_compare ) ) {
   case -1: {
     mlog(LOG_DERIVED_ERROR,"Execute: ");
     return ERROR;
  }
  case 0:
    // No online cels and/or no reference polylines
    return OK;
  case 1:
  default:
    // Continue to check
    break;
  }

  // Shrinked reference window, transformed into online
  // coordinates. Defects outside this quadrilateral will
  // not be reported.
  // Basic envelope
  Dpoint2 default_s_v_data[] = { Dpoint2(0.0,0.0), Dpoint2(1.0,0.0),
                                 Dpoint2(1.0,1.0), Dpoint2(0.0,1.0) };
  Dpolyline shrinked_ref_quadrilateral(default_s_v_data,4);
  (data_for_compare.ref_win)-
>ShrinkQuadrilateral(1.0,shrinked_ref_quadrilateral);

  // Create data for segment information, piece of info for each segment of each
  // reference polyline
  Sipdata_segments_info * segments_info = new(&w) Sipdata_segments_info;
  if ( !segments_info ) {
    mlog(LOG_ALLOC_ERROR,"Execute: Function %s: Failed to create
Sipdata_segments_info.");
    return ERROR;
  }

  //=========================================================================
  // Micro registration
  // Starting with reference polylines transformed from reference coordinates
  // to online coordinates (registration transformation), we want to compute
  // a correction so that polyline will better correspond to the online cels
  // This transformation is called polyline2onl. The inverse transformation
  // is called onl2polyline
  //=========================================================================
  Affine2dtrans polyline2onl;
  polyline2onl.SetToIdentity( w.Coordinate_System() );
  polyline2onl.SetDomainAndRange( w.Coordinate_System() );

  Affine2dtrans onl2polyline;
  onl2polyline.SetToIdentity( w.Coordinate_System() );
  onl2polyline.SetDomainAndRange( w.Coordinate_System() );

  if ( micro_reg_mode != MR_NONE ) {
    // Create segments_info from reference polylines and transform them to
online coordinates
    segments_info->SetFromPolylines( *data_for_compare.ref_polylines,
polyline2onl );

    // Set geometry for microregistration
    segments_info->SetParamsForEnhancedTest( search_width,
                                             search_width,
                                             search_width,
                                             mr_angle_tolerance );

    // Set weights for microregistration
    segments_info->SetWeight( micro_reg_segment_weight );

    Poly_compare::RetCode ret_code;
    if ( micro_reg_mode == MR_ENVELOPES ) {
      // the algorithm
      ret_code = Poly_compare::MicroRegEnvelopes( *(data_for_compare.onl_cels),
                                          segments_info->Envelopes(),
                                          polyline2onl );
    }
    else if ( micro_reg_mode == MR_ICP ) {
      segments_info->AdjustWeightByLength();
      segments_info->SetAllSegmentsAsActive();

      ret_code = Poly_compare::MicroRegICP( *segments_info,
*(data_for_compare.cel_rast),
                                    (int)search_width + 1, polyline2onl );
    }
    else {
      mlog(LOG_APPLIC_ERROR, "Execute: "
          "Function %s: Unrecognized micro registration mode (%d): ",
          Name(),micro_reg_mode);
      if ( segments_info ) {
      delete segments_info;
      }
      return ERROR;
    }


    if ( ret_code == Poly_compare::FAIL ) {
      polyline2onl.SetToIdentity( w.Coordinate_System() );
      polyline2onl.SetDomainAndRange( w.Coordinate_System() );
      mlog(LOG_WARN, "Execute: "
          "Function %s: Microreg Failed. Using identity trans for
AdjustTransformation.",
          Name());
    }

    if ( ret_code == Poly_compare::ERROR ) {
      mlog(LOG_DERIVED_ERROR,"Execute: Function %s: Microreg Failed.",
          Name());
      if ( segments_info ) {
      delete segments_info;
      }
      if ( data_for_compare.cel_rast ) {
      delete data_for_compare.cel_rast;
      }
      return ERROR;
    }

    // Adjust Self2Ref transformation according to new transformation
    onl2polyline = TransOp::Inv(polyline2onl);
    if ( !(onl2polyline.IsOk()) ) {
      mlog(LOG_APPLIC_ERROR, "Execute: "
          "Function %s: Failed to invert transformation: ", Name());
      onl2polyline.Print();
      if ( segments_info ) {
      delete segments_info;
      }
```

```
        if ( data_for_compare.cel_rast ) {
        delete data_for_compare.cel_rast;
        }
        return ERROR;
      }

      if ( !( (data_for_compare.onl_win)->AdjustSelf2Ref(onl2polyline)) ) {
        mlog(LOG_DERIVED_ERROR, "Execute: "
           "Function %s: Failed to adjust transformation: ", Name());
        if ( segments_info ) {
        delete segments_info;
        }
        if ( data_for_compare.cel_rast ) {
        delete data_for_compare.cel_rast;
        }
        return ERROR;
      }

      // Transform shrinked ref quad to new online position after
      // microreg correction.
      shrinked_ref_quadrilateral.Transform( polyline2onl );
    }

   mlog(LOG_ALL, "Execute: MicroReg shift : dx = %f dy = %f",
        polyline2onl.b1,polyline2onl.b2);

   //------------------------------------------------------------------------
   // Actual comparison
   //------------------------------------------------------------------------
   mlog(LOG_ALL, "Execute: Scan line inspection: %d pols  %d cels",
        (data_for_compare.ref_polylines)->size(),
        (data_for_compare.onl_cels)->size());

   // Create segments_info from reference polylines and transform them to online
 coordinates
   segments_info->SetFromPolylines( *data_for_compare.ref_polylines, polyline2onl
 );
   segments_info->SetParamsForEnhancedTest( defect_width_up,
                                   defect_width_down,
                                   defect_tang_width,
                                   v_sc_angle_tolerance );
   segments_info->SetAllSegmentsToBeEmptyEnvelopes();

   vector<vector< Dseg_envelope> > & ref_poly_env = segments_info->Envelopes();

 //   // clear accomulated lenght of cels inside envelopes
 //   //-------------------------------------------
 //   for ( unsigned j = 0; j < ref_poly_env.size(); ++j ) {
 //     vector<Dseg_envelope> & v_envelopes = ref_poly_env[j];
 //     for ( unsigned i = 0; i < v_envelopes.size(); ++i ) {
 //       v_envelopes[i].length_of_elements_inside = 0.0;
 //     }
 //   }


   // vector that holds CEL mismatches
   vector < const Cel_event<Cel> * > cel_mismatches;
```

```
for ( unsigned int i_compare = 0;
    i_compare < compare_mode.size();
    ++i_compare ) {     // Run over all requested comparison algorithms
  Sip_defect::Defect_type d_type = Sip_defect::VEC2CEL;
  if ( compare_mode[i_compare] == COMPARE_CELS_2_ENVELOPES ) {
    //-----------------------------------------------
    // Geometrical CELs 2 ENVELOPES test
    //-----------------------------------------------
    cel_mismatches.clear();
    if ( !CompareCelsToEnvelopes( *(data_for_compare.onl_cels),
                              ref_poly_env,
                              cel_mismatches ) ) {
    mlog(LOG_DERIVED_ERROR,"Execute: ");
    if ( segments_info ) {
      delete segments_info;
    }
    if ( data_for_compare.cel_rast ) {
      delete data_for_compare.cel_rast;
    }
    return ERROR;
    }
  }
  else if ( compare_mode[i_compare] == COMPARE_CELS_2_PIM ) {
    //-----------------------------------------------
    // CELs 2 PIM test
    //-----------------------------------------------
    if ( compare_mode.size() > 1 ) {
    // Display defect as type != VEC2CEL only if
    // more than one comparison algorithm is active.
    d_type = Sip_defect::VEC2CEL_PIM;
    }

    if ( !(data_for_compare.ref_pim_of_envelopes) ) {
    mlog(LOG_APPLIC_ERROR, "Execute: "
          "Function %s: NULL ref_pim_of_envelopes in data_for_compare"
          " %s(%d) [%d %d]x[%d %d].",
          Name(), w.Type(), w.Id(),w.X0(), w.X1(), w.Y0(), w.Y1() );
    if ( segments_info ) {
      delete segments_info;
    }
    if ( data_for_compare.cel_rast ) {
      delete data_for_compare.cel_rast;
    }
    return ERROR;
    }


    // Get PIM of envelopes, in original reference coordinates.
    const Pim<Envelope_loc_info> * main_pim
    = (data_for_compare.ref_pim_of_envelopes)->get_pim();
    double y_offset
    = (data_for_compare.ref_pim_of_envelopes)->get_y_min();
    if ( !main_pim ) {
    mlog(LOG_APPLIC_ERROR, "Execute: "
          "Function %s: NULL main_pim returned from temporary object in window"
          " %s(%d) [%d %d]x[%d %d].",
          Name(), w.Type(), w.Id(),w.X0(), w.X1(), w.Y0(), w.Y1() );
    if ( segments_info ) {
      delete segments_info;
    }
    if ( data_for_compare.cel_rast ) {
      delete data_for_compare.cel_rast;
    }
    return ERROR;
    }

    cel_mismatches.clear();
    if ( Poly_compare::CompareCelsToPIM( *(data_for_compare.onl_cels),
                                   ref_poly_env,
                                   y_offset,
                                   *main_pim,
                                   w.Self2Ref(),
                                   cel_mismatches) == Poly_compare::ERROR) {
    mlog(LOG_DERIVED_ERROR,"func_polylines_comp_aaa");
    if ( segments_info ) {
      delete segments_info;
    }
    if ( data_for_compare.cel_rast ) {
      delete data_for_compare.cel_rast;
    }
    return ERROR;
    }
    //    plog("PIM: Mismatches %d \n",cel_mismatches.size());
  }
  else if ( compare_mode[i_compare] == COMPARE_CELS_2_BITMAP ) {
    //-------------------------------------------------
    // CELs 2 bitmap test
    //-------------------------------------------------
    if ( compare_mode.size() > 1 ) {
    // Display defect as type != VEC2CEL only if
    // more than one comparison algorithm is active.
    d_type = Sip_defect::VEC2CEL_BITMAP;
    }

    double x_subpixel = 16.0;
    double y_subpixel = 1.0;
    cel_mismatches.clear();
    if ( Poly_compare::CompareCelsToBitMap( w.X0() - (2.*max_tang_width),
                                     w.Y0() - (2.*max_tang_width),
                                     w.X1() + (2.*max_tang_width),
                                     w.Y1() + (2.*max_tang_width),
                                     *(data_for_compare.onl_cels),
                                     segments_info->Envelopes(),
                                     x_subpixel, y_subpixel,
                                     cel_mismatches) == Poly_compare::ERROR) {
    mlog(LOG_DERIVED_ERROR,"func_polylines_comp_aaa");
    if ( segments_info ) {
      delete segments_info;
    }
    if ( data_for_compare.cel_rast ) {
      delete data_for_compare.cel_rast;
    }
    return ERROR;
    }
    //       plog("CompareCELS to BitMap: Mismatches %d
\n",cel_mismatches.size());
```

```
      }
      else {
        mlog(LOG_APPLIC_ERROR,"Execute: Illegal compare mode. ");
        if ( segments_info ) {
        delete segments_info;
        }
        if ( data_for_compare.cel_rast ) {
        delete data_for_compare.cel_rast;
        }
        return ERROR;
      }

      // Clip excess CELs and missing envelopes to produce real defects
      if ( !HandleDefects( cel_mismatches, ref_poly_env,
                       shrinked_ref_quadrilateral,
                       data_for_compare, d_type ) ) {
        mlog(LOG_DERIVED_ERROR,"Execute: ");
        if ( segments_info ) {
        delete segments_info;
        }
        if ( data_for_compare.cel_rast ) {
        delete data_for_compare.cel_rast;
        }
        return ERROR;
      }
    }

    //-----------------------------------------------------------------------
    // Handling of pinholes
    //-----------------------------------------------------------------------
    if ( treat_all_pinholes_as_surface_defects ) {
      // Search for pinholes in online conencted components. If found, the
      // starting CEL of the connected component will be classified as defective.
      const Sipdata_connected_components<Cel> * c_components =
        WIN_GET(Sipdata_connected_components<Cel>,w);
      if ( !c_components ) {
        mlog(LOG_APPLIC_ERROR, "Execute: Failed to get connected components from
window %s",
            w.Type());
        if ( segments_info ) {
        delete segments_info;
        }
        if ( data_for_compare.cel_rast ) {
        delete data_for_compare.cel_rast;
        }
        return ERROR;
      }

      if ( !TopologicalComparison( *(data_for_compare.ref_polylines),
                             *c_components,
                             polarity,
                             shrinked_ref_quadrilateral,
                             *(data_for_compare.sip_defects) ) ) {
        mlog(LOG_DERIVED_ERROR,"Execute: ");
        if ( segments_info ) {
        delete segments_info;
        }
        if ( data_for_compare.cel_rast ) {
```

```
        delete data_for_compare.cel_rast;
        }
        return ERROR;
      }
    }

    // Put segments_info inside window for debugging
    if ( !( w.Put(segments_info) ) ) {
      mlog(LOG_DERIVED_ERROR, "Execute: "
          "Failed to put segments_info in window");
      delete segments_info;
      return ERROR;
    }

    if ( data_for_compare.cel_rast ) {
      delete data_for_compare.cel_rast;
    }

    return OK;
  }


  //-----------------------------------------------------------------------------
  // Extract input information from window
  //-----------------------------------------------------------------------------
    int
  Func_polylines_comp_aaa::
  ExtractDataForCompare( Sipwin & w,
                       Func_polylines_comp_aaa::DataForCompare & data_for_compare )
  const
  {
    data_for_compare.sipdata_wref          = NULL;
    data_for_compare.ref_win               = NULL;
    data_for_compare.ref_polylines         = NULL;
    data_for_compare.ref_pim_of_envelopes  = NULL;

    data_for_compare.onl_win               = &w;
    data_for_compare.onl_cels              = NULL;
    data_for_compare.cel_rast              = NULL;
    data_for_compare.sip_defects           = NULL;

    Single_camera::WinGeomInclusion geom_inclusion = w.GeometricInclusionStatus();

    if ( (geom_inclusion != Single_camera::ALL_IN) &&
         (geom_inclusion != Single_camera::ALL_INSIDE) ) {
      mlog(LOG_STANDARD, "ExtractDataForCompare: "
         "Function %s: Inclusion geometry status for window %s(%d)"
         " is NOT Single_camera::ALL_IN", Name(), w.Type(), w.Id());
      //    return OK;
     }

    // Prepare Sipdata_sip_defects to collect all possible defects
    //-----------------------------------------------------------
    Sipdata_sip_defects * sip_defects = WIN_GET_NON_CONST(Sipdata_sip_defects, w);
    if ( !sip_defects  ) {
        mlog(LOG_APPLIC_ERROR, "ExtractCompareData: "
           "Function %s: Failed to find Sipdata_sip_defects for window %s(%d) (%d
%d]x[%d %d]",
            Name(), w.Type(), w.Id(),w.X0(), w.X1(), w.Y0(), w.Y1() );
        return -1;
    }
    data_for_compare.sip_defects = sip_defects;

    // Get reference data from window
    //--------------------------------------------------------------------------
    const Sipdata_winref * sipdata_wref = WIN_GET(Sipdata_winref,w);
    if ( !sipdata_wref ) {
      mlog(LOG_APPLIC_ERROR, "ExtractDataForCompare: "
          "Function %s: No Sipdata_winref element in window %s(%d) [%d %d]x[%d
%d].",
          Name(), w.Type(), w.Id(),w.X0(), w.X1(), w.Y0(), w.Y1() );
      return -1;
    }
    data_for_compare.sipdata_wref = sipdata_wref;

    // get ref window itself
    const Sipwin * ref_win = sipdata_wref->GetWin();
    if ( !ref_win ) {
      mlog(LOG_APPLIC_ERROR, "ExtractDataForCompare: "
          "Function %s: No reference window found in Sipdata_winref of window "
          "%s(%d) [%d %d]x[%d %d].", Name(), w.Type(), w.Id(),w.X0(), w.X1(),
w.Y0(), w.Y1());
      return -1;
    }
    data_for_compare.ref_win = ref_win;

    const Sipwin & ww = *ref_win;
    Dpolyline shrinked_quadrilateral;
    ww.ShrinkQuadrilateral(1.0,shrinked_quadrilateral);

    // get ref polylines
    //----------------------
    Sipdata_polylines * polylines =
      (Sipdata_polylines *)WIN_NAMED_GET(Sipdata_polylines,ww,"polylines_Cel");
    if ( polylines ) {
      if ( defect_width_up.size() < polylines->NumberOfSegId() ) {
        mlog(LOG_APPLIC_ERROR, "ExtractDataForCompare: "
            "Function %s: Number of defect thresholds (%d) is smaller than number "
            "of segment Id's in polyline objects.", Name(),
defect_width_up.size());
        return -1;
      }

      if ( polylines->IsPartialRepresentation() ) {
        mlog(LOG_APPLIC_ERROR, "ExtractDataForCompare: "
            "Function %s: Partial representation for polylines not supported
anymore.",
            Name() );
        return -1;
      }

      data_for_compare.ref_polylines = polylines;
    }

    // Get reference PIM of envelopes, if exists
    //------------------------------------------
```

```
data_for_compare.ref_pim_of_envelopes =  WIN_GET(Sipdata_pim_envelopes,ww);;

// Get Sipdata of cels from window
//---------------------------------
const Sipdata_cel_type_events<Cel> * cels
  = WIN_GET(Sipdata_cel_type_events<Cel>,w);
if ( !cels || (cels->size() < 1) ) {
  // No (online) CELs in this window
  if ( !polylines ) {
    // No reference polylines : Everything is OK.
    return 0;
  }

  // Reference polylines but no CELs - Count number of (empty) envelops
  unsigned int n_empty_envelopes = 0;
  for ( unsigned int i = 0; i < polylines->size(); ++i ) {
    const Dpolyline      & p    = (*polylines)[i];
    for ( unsigned int j= 0; j < p.size(); ++j ) {
    const Dpoint2 & pt = p[j];
    if (w.Rect().Includes_point(pt)) {
      Sip_defect d(pt.x, pt.y,Sip_defect::VEC2CEL_EMPTY_ENVELOPE);
      sip_defects->push_back(d);
      ++n_empty_envelopes;
    }
    }
  }


  if ( n_empty_envelopes > max_number_empty_envelopes  ) {
    Dpoint2 pt = w.Center();
    Sip_defect d(pt.x, pt.y, Sip_defect::VEC2CEL_EMPTY_ENVELOPES_OVERFLOW);
    sip_defects->push_back(d);
  }

  return 0;
}
else {
  // Take online CELs and create raster representation of them
  //----------------------------------------------------------
  data_for_compare.onl_cels = cels;

  if ( micro_reg_mode == MR_ICP ) {
    data_for_compare.cel_rast = new (&w) Sipdata_cel_events_raster();
    if ( !data_for_compare.cel_rast) {
    mlog(LOG_ALLOC_ERROR, "ExtractDataForCompare: "
         "Function %s: Failed to create new Sipdata_cel_events_raster"
         " in window %s(%d) [%d %d]x[%d %d].",
         Name(), w.Type(), w.Id(), w.X0(), w.X1(), w.Y0(), w.Y1());
    return -1;
    }

    if ( !( data_for_compare.cel_rast->set(cels, w.X0(), w.X1(), w.Y0(),
w.Y1()) ) ) {
    mlog(LOG_ALLOC_ERROR, "ExtractDataForCompare: "
         "Function %s: Failed on set cel raster in window %s(%d) [%d %d]x[%d
%d].",
         Name(), w.Type(), w.Id(), w.X0(), w.X1(), w.Y0(), w.Y1());
    delete(data_for_compare.cel_rast);
```

```
        data_for_compare.cel_rast = NULL;
        return -1;
        }
      }
    }


    if ( !polylines ) {
      // The reason for not having reference polylines is inability to cut
      // reference data for the specified window
      // T.B.D. Report all CELs as errors, use apropriate defect type.
      if ( ref_win->GeometricInclusionStatus() == Single_camera::ALL_OUT ) {
        mlog(LOG_WARN, "ExtractDataForCompare: "
            "Function %s: Inclusion geometry status for reference window %s(%d) "
            "is Single_camera::ALL_OUT. No reference to compare to.",
            Name(), ref_win->Type(), ref_win->Id());
      }

      // CELs exist but no reference polylines.
      // All CELs not within shrinked quadrilateral are errors.
      // Position iterator at beginning of CEls
      //------------------------------------------------------------------------
      Sipdata_cel_type_events<Cel>::const_iterator   iter(*cels);  // CELs
  iterator
      iter.Reset();
      const Cel_event<Cel>    * element;
      unsigned int n_excess_cels  = 0;
      while ( (element = iter.Next()) ) {
        double cel_point_x, cel_point_y;
        element->FirstEndpoint(cel_point_x, cel_point_y);
        if ( shrinked_quadrilateral.Includes_point(cel_point_x, cel_point_y) ) {
        Sip_defect d(cel_point_x,cel_point_y,Sip_defect::VEC2CEL);
        sip_defects->push_back(d);
        ++n_excess_cels;
        }
      }

      if ( n_excess_cels >  max_number_excess_cels  ) {
        Dpoint2 p = w.Center();
        Sip_defect d(p.x, p.y, Sip_defect::VEC2CEL_DEFECTS_OVERFLOW);
        sip_defects->push_back(d);
      }
      return 0;
    }

    // All is OK and all relevant data is extracted for further processing
    return 1;
  }


  //------------------------------------------------------------------------------
  // Compare onlyne CELs to envelopes of reference polylines
  //------------------------------------------------------------------------------
  bool
  Func_polylines_comp_aaa::
  CompareCelsToEnvelopes( const Sipdata_cel_type_events<Cel> & onl_cels,
                          vector <vector <Dseg_envelope> >   & ref_poly_env,
                          vector <const Cel_event<Cel> *>     & cel_mismatches ) const
```

```
{
  // Comparison CELs to envelopes
  //-------------------------------------------------
  bool apply_angle_criterion = (sc_angle_tolerance <= M_PI);

  // the algorithm
  cel_mismatches.clear();
  if ( Poly_compare::CompareCelsToEnvelopes( apply_angle_criterion,
                                       onl_cels,
                                       ref_poly_env,
                                       small_cel,
                                       cel_mismatches) == Poly_compare::ERROR) {
    mlog(LOG_DERIVED_ERROR,"CompareCelsToEnvelopes:");
    return false;
  }

  //  plog("Comp2Envelopes: Number of mismatches = %d\n",cel_mismatches.size());


  return true;
}


//----------------------------------------------------------------------
// Handle defects (clip and put into sip_defects)
//----------------------------------------------------------------------
bool
Func_polylines_comp_aaa::
HandleDefects(    vector <const Cel_event<Cel> *>          & cel_mismatches,
            vector<vector<Dseg_envelope> >          & ref_poly_env,
            const Dpolyline                         &
shrinked_ref_quadrilateral,
            Func_polylines_comp_aaa::DataForCompare & data_for_compare,
            Sip_defect::Defect_type d_type ) const
{
  unsigned int n_defs            = 0;
  unsigned int n_excess_cels     = 0;
  unsigned int n_empty_envelopes = 0;

  Sipdata_sip_defects * sip_defects   = data_for_compare.sip_defects;

  // run over all mismatches (CELs) not inside any envelope and decide
  // if that is an error
  // Count number of missing CELs and EXCESS CELs reported from VEC2CEL
  //---------------------------------------------------------------------
  for ( unsigned e = 0; e < cel_mismatches.size(); ++e ) {
    // write defect mark
    double cel_point_x, cel_point_y;
    cel_mismatches[e]->FirstEndpoint(cel_point_x, cel_point_y);
    if ( shrinked_ref_quadrilateral.Includes_point(cel_point_x, cel_point_y) ) {
      Sip_defect d(cel_point_x,cel_point_y,d_type);
      sip_defects->push_back(d);
      ++n_excess_cels;
      ++n_defs;
    }
  }

  if ( n_excess_cels > max_number_excess_cels  ) {
      Dpoint2 p = (data_for_compare.onl_win)->Center();
      Sip_defect d(p.x, p.y, Sip_defect::VEC2CEL_DEFECTS_OVERFLOW );
      sip_defects->push_back(d);
    }

    // Run over all empty envelopes (reference segments not covered by online
  CELs)
    // Count number of empty envelopes
    //-----------------------------------------------------------------
    // Search missings envelopes from referenece not covered by online CELs
    // run over all envelopes. For each envelope, we consider the total
    // lenght of all CELs falling inside those envelopes. If the number
    // is too small then we notify on error.
    //-------------------------------------------------------------------
    for ( unsigned i = 0; i < ref_poly_env.size(); ++i ) {
      vector <Dseg_envelope> & v_envelopes = ref_poly_env[i];
      for ( unsigned j = 0; j < v_envelopes.size(); ++j ) {
        Dseg_envelope & env = v_envelopes[j];
        if ( (env.length_of_elements_inside == 0.0) && (env.get_seg_len() > 0.0) )
  {
        // An empty envelope was detected
        double xs = env.get_seg_xs();
        double ys = env.get_seg_ys();
        if ( (data_for_compare.onl_win)->Rect().Includes_point(Dpoint2(xs, ys))) {
          Sip_defect d( xs, ys,Sip_defect::VEC2CEL_EMPTY_ENVELOPE);
          sip_defects->push_back(d);
          ++n_empty_envelopes;
          ++n_defs;
        }
        }
      }
    }

    if ( n_empty_envelopes > max_number_empty_envelopes  ) {
      Dpoint2 p = (data_for_compare.onl_win)->Center();
      Sip_defect d(p.x, p.y, Sip_defect::VEC2CEL_EMPTY_ENVELOPES_OVERFLOW);
      sip_defects->push_back(d);
    }

    if ( n_defs > 0 ) {
      mlog(LOG_STANDARD, "poly_compare defects %3d OUT OF BAND DEFECTS. %d pols
  %d cels",
          n_defs,
          (data_for_compare.ref_polylines)->size(),
          (data_for_compare.onl_cels)->size());
    }

    return true;
  }


  //-------------------------------------------------------------------
  // Topological comparison between reference and online. Currently
  // applied only to pinholes. This is a very primitive version for
  // giving quick answer for false alarm that arise because we can not
  // classify pinholes and test them agains a surface criterion.
  //------------------------------------------------------------------------
  bool
```

```
Func_polylines_comp_aaa::
TopologicalComparison( const Sipdata_polylines & ref_polylines,
                       const Sipdata_connected_components<Cel> & onl_c_components,
                       int polarity,
                       const Dpolyline & shrinked_ref_quadrilateral,
                       Sipdata_sip_defects & sip_defects ) const
{
  for (unsigned i = 0; i < ref_polylines.size(); ++i) {
    if ( ref_polylines[i].IsClosed() ) {
      double sa = ref_polylines[i].SignedArea2();
      bool reverse_polarity = ( ((polarity == 1) && (sa > 0)) ||
                                ((polarity == 0) && (sa < 0)) );
      if ( reverse_polarity ) {
      // The reference data contains a pinhole; No further testing is done
      // A real topological comparison would try to match betweeb reference
      // polylines and online connected components and will be able to
      // report on extra pinholes that do not match holes from the reference.
      //    fprintf(stderr," PINHOLE IN REF \n");
      return true;
      }
    }
  }

  // reference data does not contain a pinhole.
  for (unsigned i = 0; i < onl_c_components.size(); ++i) {
    if ( (onl_c_components[i].size() > 0) && (onl_c_components[i].IsClosed()) )
{
      // 1. polarity stuff
      // positive polygon surrounds white area (negative for this polarity)
      // nagative polygon surrounds black area (positive for this polarity)
      double sa = onl_c_components[i].SignedArea2();
      bool reverse_polarity = ( ((polarity == 1) && (sa > 0)) ||
                                ((polarity == 0) && (sa < 0)) );

      if( reverse_polarity ) {
      // Deal with polygons with reverse polarity (pinholes).
      //    fprintf(stderr," PINHOLE found (%d)\n",i);
      const Connected_component<Cel> & chain = onl_c_components[i];

      bool all_cels_outside_window = true;
      for (unsigned i = 0; i < chain.size(); ++i) {
        if ( shrinked_ref_quadrilateral.Includes_point(chain[i].x, chain[i].y) )
{
          all_cels_outside_window = false;
          break;
        }
      }

      // A pinhole is a defect only if it is resonably far away from the
      // window's shrinked quadrilateral.
      if ( !all_cels_outside_window ) {
        for (unsigned i = 0; i < chain.size(); ++i) {
          Sip_defect d(chain[i].x, chain[i].y, Sip_defect::VEC2CEL);
          sip_defects.push_back(d);
        }
      }
      }
    }
    }

    return true;
  }



/*
    // Get connected components from online window
    //--------------------------------------------------------------------------
    const Sipdata_connected_components<Cel> * c_components;
    c_components = WIN_GET(Sipdata_connected_components<Cel>,w);
    bool connected_inspection = c_components && (c_components->size() > 0);
    if ( !connected_inspection && partial_representation )
    {
      // in the partial_representation case we allow only connected
      // component inspection.
      double xd = (double)((w.Trigger()).x);
      double yd = (double)((w.Trigger()).y);
      Sip_defect d(xd,yd,"MISSING_DATA_FOR_COMPUTATION","no connected
components");
      sip_defects->push_back(d);
      polylines->DestroyEnvelopes();
      return DEFECTS_FOUND;
    }

    if ( connected_inspection )
    {
      mlog(LOG_STANDARD,"Connected components inspection");

      // connected component inspection.
      // Applied only in full or partial  representation mode.
      //------------------------------------------------------------------------
      vector<unsigned> mismatches;
      if (partial_representation)
      {
        // Partial representation of cels in window as polylines.
        for ( unsigned j = 0; j < polylines->size(); ++j )
        {
        const vector<Dseg_envelope> & v_envelopes = (*polylines).Envelopes()[j];
        unsigned number_of_matches = 0;
        for ( unsigned i = 0; i < c_components->size(); ++i )
        {
          const Connected_component<Cel> & c = (*c_components)[i];
          pair<unsigned,unsigned> match;
          bool is_match = Poly_compare::Match( c,
                                   ww.Quadrilateral(),
                                   v_envelopes,
                                   match);

          if ( is_match )
          {
            number_of_matches++;
            mismatches.erase(mismatches.begin(),mismatches.end());
            Poly_compare::FindMismatches( c,
                                 shrinked_quadrilateral,
                                 polylines->Envelopes(),
                         mismatches );

          for ( unsigned e = 0; e < mismatches.size(); ++e )
          {
            // write defect mark
            unsigned mindex = mismatches[e];
            if ( shrinked_quadrilateral.Includes_point(c[mindex].x,
  c[mindex].y))
            {
            Sip_defect d(c[mindex].x, c[mindex].y,
                       "DTYPE_POINT_MISMATCH","CEL2VEC");
            sip_defects->push_back(d);
            n_defs++;
            }
          }
          //          break;
        }
      }

      if ( number_of_matches != 1 )
      {
        // in the partial_representation case we allow only connected
        // component inspection.
        double xd = (double)((w.Trigger()).x);
        double yd = (double)((w.Trigger()).y);
        Sip_defect d(xd,yd,"DTYPE_MISSING_MATERIAL_IN_WINDOW",
                   "no connected components");
        sip_defects->push_back(d);
        polylines->DestroyEnvelopes();
        return DEFECTS_FOUND;
      }
      }
    } // end of partial representation connected inspection
    else
    {
      // Full representation of cels in window as polylines.
      for ( unsigned i = 0; i < c_components->size(); ++i )
      {
      const Connected_component<Cel> & c = (*c_components)[i];
      mismatches.erase(mismatches.begin(),mismatches.end());
      Poly_compare::FindMismatches( c,
                         shrinked_quadrilateral,
                         polylines->Envelopes(),
                         mismatches );

      for ( unsigned e = 0; e < mismatches.size(); ++e )
      {
        // write defect mark
        unsigned mindex = mismatches[e];
        if ( shrinked_quadrilateral.Includes_point(c[mindex].x, c[mindex].y) )
        {
          Sip_defect d(c[mindex].x, c[mindex].y,
                   "DTYPE_POINT_MISMATCH","CEL2VEC");
          sip_defects->push_back(d);
          n_defs++;
        }
      }
      }
      } // end of full representation connected inspection
    } // end of connected inspection
   else
  */



  /*
   *  $Log: func_polylines_comp_aaa.C,v $
   *  Revision 1.63  2001/08/29 10:31:08  shmulik
   *  Support for enhanced (fastes) microregistration and
   *  comparison algorithms
   *
   *  Code Review by
   *
   *  Revision 1.62  2001/01/23 14:54:07  shmulik
   *  Detecting pinholes as part of CEL2VEC comparison.
   *
   *  Code Review by
   *
   *  Revision 1.61  2000/07/18 08:00:41  shmulik
   *  Automatic computation of tangential width as the average between
   *  the up and down widthes.
   *
   *  Code Review by Meir
   *
   *  Revision 1.60  2000/07/17 06:34:48  shmulik
   *  Supporting the possibility for unsymmetric envelope widhtes
   *  (different sizes for nick and protrusion testing)
   *
   *  Code Review by Meir
   *
   *  Revision 1.59  2000/07/16 13:48:46  shmulik
   *  Preparation for setting different widthes for the up and down sides of
   *  an oriented envelope.
   *
   *  Code Review by Meir
   *
   *  Revision 1.58  1999/12/23 08:48:16  shmulik
   *  changing return value of method Type() from string to const char *
   *
   *  Code Review by Meir
   *
   *  Revision 1.57  1999/12/19 13:04:29  sharond
   *  Add log message
   *
   *  Code Review by
   *
   *  Revision 1.56  1999/12/16 12:42:47  meirm
   *  haqndling of missing shape
   *
   *  Code Review by
   *
   *  Revision 1.55  1999/10/24 14:26:44  joseph-w
   *  changing to return code instead of true/false
   *
   *  Code Review by
   *
```

```
 *  Revision 1.54  1999/10/14 13:27:10  meirm
 *  improving includes
 *
 *  Code Review by
 *
 *  Revision 1.53  1999/09/26 16:08:25  joseph-w
 *  changing the order of including the x.H file to first line of x.C
 *
 *  Revision 1.52  1999/09/23 10:02:09  meirm
 *  *** empty log message ***
 *
 *  Revision 1.51  1999/08/23 07:22:50  sharond
 *  Rearranging balls path & Adding support for iballs path
 *
 *  Revision 1.50  1999/08/12 13:12:59  joseph-w
 *  New interface for functions
 *
 *  Revision 1.49  1999/08/10 09:35:06  shmulik
 *  Adding weight for micro registration calculations.
 *
 *  Revision 1.48  1999/08/08 08:27:21  sharond
 *  Messages cleanup
 *
 *  Revision 1.47  1999/07/21 12:45:42  shmulik
 *  Continuing with identity correction after micro registration fail.
 *
 *  Revision 1.46  1999/07/14 12:56:02  meirm
 *  *** empty log message ***
 *
 *  Revision 1.45  1999/07/07 15:44:57  meirm
 *  *** empty log message ***
 *
 *  Revision 1.44  1999/07/07 12:56:31  sharond
 *  New coordinate model
 *
 *  Revision 1.43  1999/07/06 12:23:56  meirm
 *  *** empty log message ***
 *
 *  Revision 1.42  1999/07/06 05:41:29  meirm
 *  *** empty log message ***
 *
 *  Revision 1.41  1999/06/23 12:04:01  meirm
 *  *** empty log message ***
 *
 *  Revision 1.40  1999/06/16 06:18:34  meirm
 *  cosmetic
 *
 *  Revision 1.39  1999/03/30 06:52:08  meirm
 *  changes in log messages
 *
 *  Revision 1.38  1999/03/09 08:21:24  meirm
 *  changing to win_named_get of sipdata_polylijnes (Cel)
 *
 *  Revision 1.37  1999/03/03 14:31:36  sharond
 *  Avoid unnecessary Print()
 *
 *  Revision 1.36  1999/03/01 14:18:39  shmulik
 *  Improved handling of degenerate polylines (two vertices that are
 *  too close together) and degenerate envelopes (zero length)
 *
 *  Revision 1.35  1999/03/01 13:48:35  sharond
 *  Arrange logging messages
 *
 *  Revision 1.34  1999/02/28 18:40:27  shmulik
 *  Adding global defects when two many excess CELs outside
 *  envelopes or empty envelopes without CELs are recorded.
 *
 *  Revision 1.32  1999/02/16 13:35:06  joseph-w
 *  transformation fixes
 *
 *  Revision 1.30  1999/02/11 14:07:58  sharond
 *  *** empty log message ***
 *
 *  Revision 1.29  1999/01/12 10:27:50  sharond
 *  Revision of the coordinate system
 *
 *  Revision 1.28  1998/12/21 06:41:50  meirm
 *  *** empty log message ***
 *
 *  Revision 1.27  1998/12/16 12:25:07  shmulik
 *  adding strict type checking for coordinate type
 *
 *  Revision 1.26  1998/09/06 07:07:52  meirm
 *  correction of vector <vector <dseg envelopes > > bug
 *
 *  Revision 1.25  1998/08/25 09:08:17  meirm
 *  *** empty log message ***
 *
 *  Revision 1.24  1998/08/11 12:27:03  sharond
 *  Camera Model + Sip General Data modifications
 *
 *  Revision 1.23  1998/06/30 14:00:38  meirm
 *  *** empty log message ***
 *
 *  Revision 1.22  1998/06/10 09:59:56  shmulik
 *  Improvements in CEL2VEC caclulations
 *
 *  Revision 1.21  1998/05/11 17:14:30  shmulik
 *  Adding changes for "missing" detection (envelopes from
 *  reference without any onl;ine CELs inside )
 *
 *  Revision 1.20  1998/04/08 07:08:36  eyalk
 *  converting to string, to ansi conventions, new factory, etc ...
 *
 *  Revision 1.19  1998/03/30 04:59:50  meirm
 *  *** empty log message ***
 *
 *  Revision 1.18  1998/02/19 13:55:48  eyalk
 *  *** empty log message ***
 *
 *  Revision 1.17  1998/02/19 07:24:08  eyalk
 *  updates of new compiler
 *
 *  Revision 1.16  1998/02/10 09:14:06  meirm
 *  *** empty log message ***
 *
 *  Revision 1.15  1998/02/03 06:10:57  shmulik
 *  Algoriyhm improvements
 *
 *  Revision 1.14  1998/01/15 10:27:34  shmulik
 *  *** empty log message ***
 *
 *  Revision 1.13  1998/01/06 14:11:44  shmulik
 *  Replacing all LOG_CONFIG log messages by LOG_APPLIC
 *
 *  Revision 1.12  1998/01/01 09:58:58  shmulik
 *  Integrating new logger into the SIP
 *
 *  Revision 1.11  1997/12/29 07:44:18  yulia
 *  Adding NEW logger
 *
 *  Revision 1.10  1997/12/14 14:29:58  shmulik
 *  adding support for templated version of CEL data structures
 *
 *  Revision 1.9  1997/11/27 08:15:25  shmulik
 *  *** empty mlog message ***
 *
 *  Revision 1.8  1997/11/09 10:31:57  shmulik
 *  *** empty mlog message ***
 *
 *  Revision 1.7  1997/11/05 14:51:26  meirm
 *  Adding support for multiply envelops/critical area handling
 *
 *  Revision 1.6  1997/11/04 09:17:51  shmulik
 *  *** empty mlog message ***
 *
 *  Revision 1.5  1997/10/20 09:45:41  meirm
 *  *** empty mlog message ***
 *
 *  Revision 1.4  1997/10/13 14:18:01  shmulik
 *  adding support for sipdata_sip_defects
 *
 *  Revision 1.3  1997/10/13 07:22:58  shmulik
 *  continuing improvements
 *
 *  Revision 1.2  1997/10/12 07:52:45  shmulik
 *  *** empty mlog message ***
 *
 *  Revision 1.1  1997/10/05 09:34:05  meirm
 *  *** empty mlog message ***
 *
 *
 */
```